# 实验10 实现转移指令的CPU设计实验

# 一、实验目的

- 掌握RISC-V 转移指令的数据通路设计；
- 掌握掌握指令流和数据流的控制方法；
- 学习依据新增指令，修改多周期CPU系统结构的方法
- 具备连接各模块构建整机的能力
- 实现转移类指令beq、jal、jalr的功能

# 二、实验内容与原理

- **实验内容：**在前述构建并实现了运算类指令（R型和I型）、传送指令（U型lui指令）、访存指令（I型和S型）的RISC-V CPU基础上，新增三条转移类指令beq、jal、jalr；
- 在实验9的基础上，添加PC0寄存器、相对转移的地址加法器和PC写入的多路选择器，并与各个逻辑模块正确连接
- 修改寄存器堆的写入数据多路选择器、修改二级译码及控制单元，实现新的目标指令集功能
- 编写测试程序，检验CPU是否能正确执行基于新目标指令集的程序

1. 新增目标指令
2. 转移指令的数据通路与执行过程
3. 控制单元CU
4. 测试程序

# 1、新增目标指令

## B型分支指令beq编码及功能

| 位数 | 7位 | 5位 | 5位 | 3位 | 5位 | 7位 | 指令功能 |
|---|---|---|---|---|---|---|---|
| B型指令字段 | imm[12,10:5] | rs2 | rs1 | funct3 | imm[4:1,11] | opcode | |
| 汇编指令 | 机器指令编码 | | | | | | |
| beq rs1,rs2,offset12 | offset7 | rs2 | rs1 | 000 | offset5 | 1100011 | if(rs1=rs2) PC+imm32→PC |

- B型分支指令的共同特征：opcode=1100011
- beq：比较rs1和rs2寄存器内容，相等则转移
- 相对转移方式：转移地址=PC+imm32
- imm32=SE32({offset12[12:1]},0}

# 1、新增目标指令

## J型跳转并链接指令jal编码及功能

| 位数 | 20位 | 5位 | 7位 | 指令功能 |
|---|---|---|---|---|
| J型指令字段 | imm[20,10:1,11,19:12] | rd | opcode | |
| 汇编指令 | 机器指令编码 | | | |
| jal rd,offset20 | offset20 | rd | 1101111 | PC+4→rd,<br>PC+imm32→PC |

- 链接：PC+4→rd
- 相对转移方式：转移地址=PC+imm32
- imm32=SE32({offset20[20:1]},0}

# 1、新增目标指令

## I型跳转并链接指令jalr编码及功能

| 位数 | 12位 | 5位 | 3位 | 5位 | 7位 | 指令功能 |
|---|---|---|---|---|---|---|
| I型指令字段 | offset12 | rs1 | funct3 | rd | opcode | |
| 汇编指令 | 机器指令编码 | | | | | |
| jalr rd,offset12(rs1) | offset12 | rs1 | 000 | rd | 1100111 | PC+4→rd,<br>rs1+imm32→PC |

- 链接：PC+4→rd
- 相对寄存器转移方式：转移地址=rs1+imm32
- imm32=SE32(offset12}

# 2、转移指令的数据通路与执行过程



- （1）添加新的部件
  - PC多路数据选择器
  - 相对转移地址加法器
  - PC0寄存器
- （2）修改rd的写入数据W_data多路选择器
- （3）转移指令的数据通路
- （4）转移指令的执行过程

# （1）添加新的部件

- PC多路数据选择器：三选一
  - 取指令：PC+4→PC
    - 实现加法方式：专用的自增4加法器
    - 来源1：PC自增4加法器输出
  - beq、jal：PC+imm32→PC
    - 实现加法方式：增设相对地址加法器
    - 来源2：相对地址加法器的输出
  - jalr：rs1+imm32→PC
    - 实现加法方式： ALU，结果在F中
    - 来源3：F
- 相对地址加法器：
  - 实现PC+imm32
  - PC：本条指令地址
  - 当前PC内容：已经+4，指向下条指令地址
  - 增设PC0寄存器：保存+4前的本条指令地址
- PC0寄存器：
  - 保存本条指令地址
  - 取指令周期完成PC→PC0

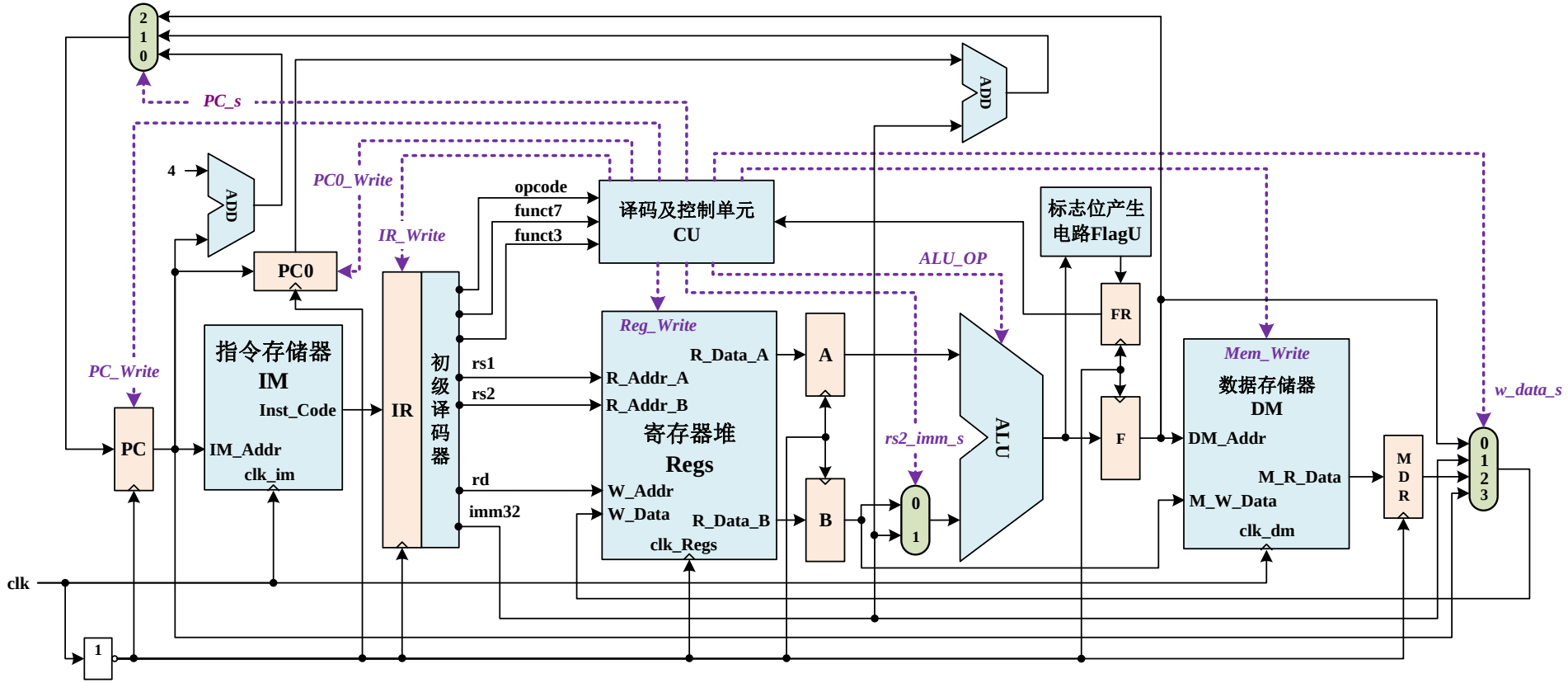
# （2）修改W_data多路选择器

- jal、jalr指令：PC+4→rd
  - 写入rd的数据W_data来源多了一个PC+4，其实是PC（已经在取指令周期+4）

- W_Data四选一多路选择器：w_data_s[1:0]选择信号
  - w_data_s=00：W_Data=F，用于R型指令和I型运算指令
  - w_data_s=01：W_Data=imm32，用于U型指令lui
  - w_data_s=10：W_Data=MDR，用于I型访存指令lw
  - w_data_s=11：W_Data=PC，用于J型指令jal、I型指令jalr

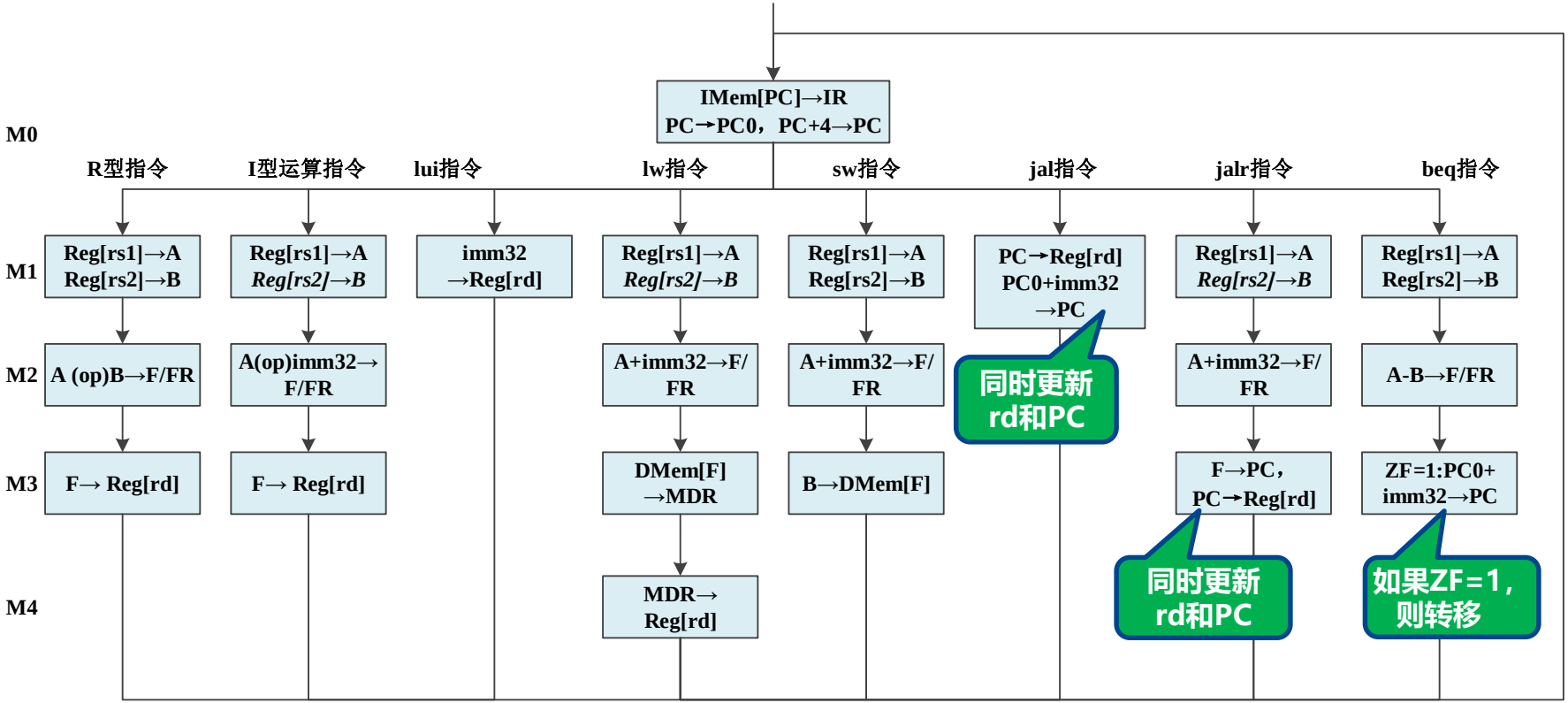
- PC三选一多路选择器：PC_s[1:0]选择信号
  - PC_s=00：PC自增加法器的输出→PC，用于取指令周期实现PC+4→PC
  - PC_s=01：相对转移地址加法器的输出→PC，用于jal指令、ZF=1的beq指令实现PC0+imm32→PC
  - PC_s=10：F→PC，用于jalr指令实现rs1+imm32→PC

# （3）转移指令的数据通路


2
1
0
PC_s
ADD
4
ADD
PC0_Write
IR_Write
opcode
funct7
funct3
译码及控制单元
CU
标志位产生
电路FlagU
ALU_OP
PC0
FR
PC_Write
指令存储器
IM
Inst_Code
IM_Addr
clk_im
IR
初级译码器
rs1
rs2
rd
imm32
Reg_Write
R_Data_A
R_Addr_A
R_Addr_B
寄存器堆
Regs
W_Addr
W_Data
R_Data_B
clk_Regs
A
B
rs2_imm_s
0
1
ALU
F
Mem_Write
数据存储器
DM
DM_Addr
M_R_Data
M_W_Data
clk_dm
M
D
R
w_data_s
0
1
2
3
PC
clk
1

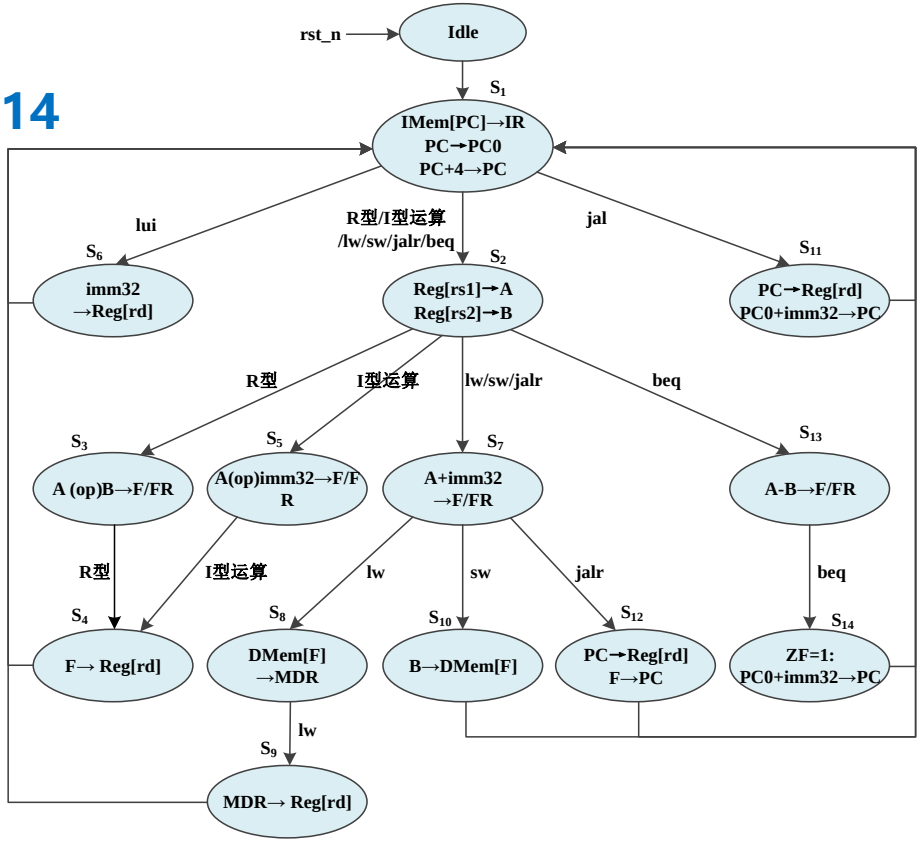
# (4) 转移指令的执行过程




M0
IMem[PC]→IR
PC→PC0，PC+4→PC
R型指令
I型运算指令
lui指令
lw指令
sw指令
jal指令
jalr指令
beq指令
M1
Reg[rs1]→A
Reg[rs2]→B
Reg[rs1]→A
Reg[rs2]→B
imm32
→Reg[rd]
Reg[rs1]→A
Reg[rs2]→B
Reg[rs1]→A
Reg[rs2]→B
PC→Reg[rd]
PC0+imm32
→PC
Reg[rs1]→A
Reg[rs2]→B
Reg[rs1]→A
Reg[rs2]→B
M2
A (op)B→F/FR
A(op)imm32→
F/FR
A+imm32→F/
FR
A+imm32→F/
FR
同时更新
rd和PC
A+imm32→F/
FR
A-B→F/FR
M3
F→ Reg[rd]
F→ Reg[rd]
DMem[F]
→MDR
B→DMem[F]
F→PC，
PC→Reg[rd]
ZF=1:PC0+
imm32→PC
同时更新
rd和PC
如果ZF=1，
则转移
M4
MDR→
Reg[rd]

# 3、控制单元CU

- （1）指令执行的状态转移图
- CU的状态转移图多了4个状态S11~S14
  - S11状态完成jal的转移并链接
  - S12执行jalr的转移并链接操作
  - S13执行beq指令的减法比较
  - S14执行beq指令的有条件转移操作




beq指令的状态转移流程是：

S1→S2→S13→S14→S1

jal指令的状态转移流程是：S1→S11→S;

jalr指令的状态转移流程是：

S1→S2→S7→S12→S1;

# (2) 状态输出的控制信号



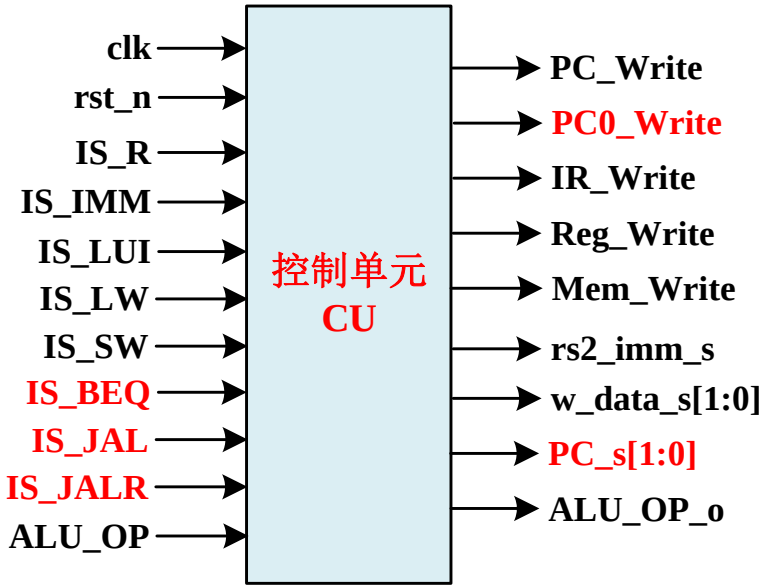
| 状态 / 控制信号 | $S_1$ | $S_2$ | $S_3$ | $S_4$ | $S_5$ | $S_6$ | $S_7$ | $S_8$ | $S_9$ | $S_{10}$ | $S_{11}$ | $S_{12}$ | $S_{13}$ | $S_{14}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PC_Write | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | ZF |
| PC0_Write | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| IR_Write | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Reg_Write | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| Mem_Write | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| ALU_OP | — | — | **** | — | **** | — | 0000 | — | — | — | — | — | 1000 | — |
| rs2_imm_s | — | — | 0 | — | 1 | — | 1 | — | — | — | — | — | 0 | — |
| w_data_s[1:0] | — | — | — | 00 | — | 01 | — | — | 10 | — | 11 | 11 | — | — |
| PC_s[1:0] | 00 | — | — | — | — | — | — | — | — | — | 01 | 10 | — | 01 |

- beq的状态S14：根据状态S13执行减法运算产生的ZF标志，来执行有条件转移
- ZF=1表示rs1-rs2=0，即rs1=rs2，则转移，这时PC_Write=1，PC_s=01
- ZF=0表示rs1-rs2≠0，即rs1≠rs2，则不转移，这时PC_Write=0，PC_s=xx（无关）
- 分析它们的逻辑，综合起来就可以用ZF产生PC_Write，即PC_Write=ZF，PC_s取01即可

# （2）状态输出的控制信号

- 修改有限状态机的状态输出函数，修改发送的控制信号，就实现了能控制25条指令执行过程的控制单元CU。

- 新的CU模块：
- 输入信号：多了三条指令译码信号IS_BEQ、IS_JAL、IS_JALR，来自二级译码模块ID2；
- 输出信号：多了PC0_Write、PC_s[1:0]



# 4、测试程序范例



```
main:
        addi   a0, zero, 0x10;     #a0=0000_0010H，数据区域（数组）首址
        ori    a1, zero, 3         #a1=0000_0003H，累加的数据个数
        xori   a2, zero, 0x30;     #a2=0000_0030H，累加和存放的单元
        jal    BankSum             #子程序调用
        lw     s0, 0(a2)           #读出累加和
BankSum:
        add    t0,  a0,    zero;   #t0=数据区域首址
        or     t1,  a1,    zero;   #t1=计数器，初始为累加的数据个数
        and    t2,  zero,  zero;   #t2=累加和，初始清零
L:      lw     t3,  0(t0);         #t3=取出数据
        add    t2,  t2,    t3      #累加
        addi   t0,  t0,    4;      #移动数据区指针
        addi   t1,  t1,    -1;     #计数器-1
        beq    t1,  zero,  exit;   #计数值=0，累加完成，退出循环
        j      L                   #计数值≠0，继续累加，跳转至循环体首部
exit: sw    t2, 0(a2);             #累加和，存到指定单元
        jr     ra                  #子程序返回
```

为了测试更多的指令，上述程序中将通常使用的addi指令，在不改变执行结果的情况下，特意使用了ori、xori、or、and等指令替代addi。

将测试程序对应的机器指令代码填入COE文件，用来初始化指令存储器。而数据存储器也通过COE文件初始化为待测试的数据。

# 三、实验要求



1. 编写用于测试的汇编程序RIUSJB_test.s，可以自行设计，也可以采用前述范例程序；将其翻译成机器指令代码，写入RIUSJB_test.coe文件备用。
2. 将实验9的工程另存为一个新工程，用RIUSJB_test.coe重新生成指令存储器模块，用测试数据初始化数据存储器。
3. 设置PC多路选择器，新建PC0寄存器，添加相对转移地址加法器，改造W_Data的多路数据选择器为四选一数据选择器；修改二级指令译码模块ID2和控制单元CU；将它们与其他模块进行正确的连接，构建RIUSJB_CPU模块
4. 对RIUSJB_CPU模块进行仿真，在rst_n之后，每来一个clk，观察指令执行的每一步骤（状态）是否符合预期，确保RIUSJB_CPU能正确执行目标指令集上的测试程序。

# 三、实验要求

5. 针对使用的实验板卡，设计RIUSJB_CPU的板级验证实验方案，编写顶层测试模块，要求至少能观察PC、IR、W_Data、MDR、标志位。可参照实验9方案设计。
6. 复位后，按clk按键，会按步骤执行指令，验证每条指令需要几个周期执行完，执行结果是否正确，将实验结果记录下来。

| 序号 | PC | IR | 汇编指令 | 执行结果 | 下条指令地址 |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

# 三、实验要求

7. 撰写实验报告，重点内容包括：

1）对仿真结果进行分析；描述你设计的板级验证实验方案、模块结构与连接；说明你的板级操作过程；记录板级实验结果。

2）针对以下有待验证的实验问题，**给出证据与分析，得到有效结论**：

①beq指令能正确做出数据比较，并进行分支；

②jal指令能完成子程序的调用，也能完成无条件的跳转；

③jalr指令能完成子程序返回；

④测试程序执行过程中，循环执行次数及PC变化符合预期。

3）请力所能及回答或实践本实验的“思考与探索”部分。

# 四、实验步骤

1. 编写用于测试的汇编程序RIUSJB_test.s，可以自行设计，也可以采用前述范例程序；
2. 使用汇编器和反汇编器，将其翻译成机器指令代码，写入RIUSJB_test.coe文件备用；
3. 新建数据存储器的COE文件RIUSJB_data.coe备用，初始化为想要测试的数据
4. 将实验9的工程另存为一个新工程；
5. 用RIUSJB_test.coe重新初始化指令存储器的IP核模块IM，用RIUSJB_data.coe重新初始化数据存储器；
6. 新建PC0寄存器，与PC正确连接；

# 四、实验步骤

7. 新建PC的三选一多路选择器、相对转移地址加法器，并与PC、PC0、初级译码器等其他部件进行正确连接；
8. 改造W_Data的多路数据选择器为四选一数据选择器；
9. 修改二级指令译码模块ID2，添加对beq、jal和jalr指令的译码；
10. 修改控制单元CU的有限状态机，添加beq、jal和jalr指令的状态及其控制；
11. 将各模块正确连接，构建RIUSJB_CPU模块；
12. 编写激励仿真代码，对RIUSJB_CPU模块进行仿真测试，首先rst_n=0，然后置1，之后，周期性地产生clk，观察输出信号，分析指令执行的每一步骤（状态）是否符合预期。确保RIUSJB_CPU能正确执行目标指令集上的程序。

# 四、实验步骤

13. 依据实际板卡情况，设计RIUSJB_CPU模块的板级验证实验方案，然后据此编写一个顶层测试模块。按照要求至少能观察PC、IR、MDR、写入寄存器堆的值W_Data。
14. 新建**管脚约束文件**，进行相应的**引脚配置**；
15. 生成*.bit文件，下载到实验设备的**FPGA芯片**中。
16. 板级实验：按照你所设计的实验方案，操作输入设备、观察输出设备，预期的**验证操作**如下：

1）按rst_n按键，将PC和IR清零，接下来将从0号单元开始执行指令；

2）按动时钟键clk，每按一下，指令就执行一步，观察PC和IR的值，判断PC是否能够正确的+4或者跳转；观察各部件值，判断其他指令执行结果是否正确。

# 五、思考与探索（至少完成1道）



1. 你的CPU能否正确执行新增的三条转移指令？以你的测试程序为例，举例说明。
2. 测试范例程序中，j L指令是伪指令，观察反汇编文件，它是用什么指令实现的？如果是用jalr实现，请用jal指令实现相同的功能；如果是用jal指令实现的，请改用jalr指令实现。
3. 现在再次思考添加U型指令auipc rd，imm20，其功能是PC+{imm20,12{0}}→rd。请问：能否使用相对转移地址专用加法器，来实现这个加法？如果可以，请问还要如何修改数据通路？尝试写出auipc的指令执行流程图，修改状态图和CU部件，实现该指令。
4. 尝试添加B型分支指令的另外5条，说说你是如何通过标志位来判断转移条件是否满足。
5. 说说你在实验中碰到了哪些问题，你是如何解决的？